CLAIMS

1. One or more computer-readable media having stored thereon a plurality of instructions that, when executed by one or more processors of a computer, causes the one or more processors to perform acts including:

allowing operation of the computer to begin based on untrusted code;

loading, under control of the untrusted code, a trusted core into memory;

preventing each of one or more central processing units and each of one or more bus masters in the computer from accessing the memory;

resetting each of the one or more central processing units;

allowing one central processing unit to access the memory and execute trusted core initialization code to initialize the trusted core; and

after execution of the trusted core has been initialized, allowing any other central processing units and any bus masters in the computer to access the memory.

2. One or more computer-readable media as recited in claim 1, wherein the one or more processors comprise one or more controllers of one or more memory controllers.

3. One or more computer-readable media as recited in claim 2, wherein the one or more memory controllers are distributed among the one or more central processing units.

4. One or more computer-readable media as recited in claim 2, wherein the plurality of instructions comprise microcode to be executed by the one or more memory controllers.

5. One or more computer-readable media as recited in claim 1, wherein the untrusted code includes code from a basic input output system (BIOS) and code from a plurality of option read only memories (ROMs).

6. One or more computer-readable media as recited in claim 1, wherein the preventing comprises preventing each of the one or more central processing units and each of the one or more bus masters from accessing the memory in response to an initialize trusted core command received from one of the one or more central processing units.

7. One or more computer-readable media as recited in claim 1, wherein the loading the trusted core comprises copying different portions of the trusted core from a plurality of different sources.

8. One or more computer-readable media as recited in claim 1, wherein the loading the trusted core comprises copying different parts of the trusted core from one or more sources and combining the different parts to assemble the trusted core.

9. One or more computer-readable media as recited in claim 1, wherein combining the different parts comprises exclusive-ORing bits of the different parts.

10. One or more computer-readable media as recited in claim 1, wherein the loading the trusted core comprises copying at least a portion of the trusted core from a local mass storage device into the memory.

11. One or more computer-readable media as recited in claim 1, wherein the loading the trusted core comprises copying at least a portion of the trusted core from a remote device into the memory.

12. One or more computer-readable media as recited in claim 1, wherein the loading the trusted core comprises copying at least a portion of the trusted core from a chip of the computer.

13. One or more computer-readable media as recited in claim 1, wherein the preventing comprises ignoring all requests for access to the memory from the one or more central processing units and one or more bus masters.

14. One or more computer-readable media as recited in claim 1, wherein the plurality of instructions further cause the one or more processors to perform acts including:

extracting a cryptographic measure of the trusted core in the memory; and

storing the extracted cryptographic measure.

15. One or more computer-readable media as recited in claim 14, wherein the plurality of instructions further cause the one or more processors to perform acts including:

resetting a cryptographic processor;

requesting the cryptographic processor to extract the cryptographic measure; and

receiving the extracted cryptographic measure from the cryptographic processor.

16. One or more computer-readable media as recited in claim 1, wherein the resetting each of the one or more central processing units comprises asserting a processor bus reset signal to each of the one or more central processing units.

17. One or more computer-readable media as recited in claim 1, wherein the plurality of instructions further cause the one or more processors to perform acts including:

mapping a central processing unit reset vector to an initialization vector;

receiving a read request corresponding to the central processing unit reset vector from the one central processing unit;

returning, in response to the read request, the initialization vector to the one central processing unit; and

allowing the one central processing unit to access the memory beginning with the initialization vector.

18. One or more computer-readable media as recited in claim 17, wherein the initialization vector is an address within the trusted code in the memory.

19. One or more computer-readable media as recited in claim 17, wherein the plurality of instructions further cause the one or more processors to perform acts including:

re-mapping the central processing unit reset vector to an additional central processing unit start vector after returning the initialization vector to the one central processing unit; and

returning, in response to any other read request corresponding to the central processing unit reset vector from another central processing unit, the additional central processing unit start vector.

20. One or more computer-readable media as recited in claim 19, wherein the initialization vector is an address within the trusted code in the memory and wherein the additional central processing unit start vector and the initialization vector are different addresses within the trusted code in the memory.

21. One or more computer-readable media as recited in claim 19, wherein both the initialization vector and the additional central processing unit start vector are obtained from the trusted core.

22. One or more computer-readable media as recited in claim 1, wherein the plurality of instructions further cause the one or more processors to perform acts including loading microcode from the trusted core in memory into the one central processing unit after resetting the central processing unit.

23. A method comprising:

booting, based on untrustworthy code, a computer;

loading a trusted core into memory; and

initiating secure execution of the trusted core.

24. A method as recited in claim 23, further comprising:

allowing execution of the trusted core to terminate; and

re-initiating secure execution of the trusted core without re-booting the computer.

25. A method as recited in claim 23, further comprising:

allowing execution of the trusted core to terminate;

loading another trusted core into memory; and

initiating secure execution of the other trusted core.

26. A method as recited in claim 25, wherein the trusted core and the other trusted core are different versions of the same trusted core.

27. A method as recited in claim 23, wherein the initiating comprises initiating secure execution of the trusted core in response to an initialize trusted core command received from one of the one or more central processing units.

28. A method as recited in claim 23, wherein the initiating comprises initiating secure execution of the trusted core without requiring any additional bus transactions to be supported by processors in the computer.

29. A method as recited in claim 23, wherein the initiating secure execution of the trusted core comprises:

preventing each of one or more central processing units in the computer from accessing the memory;

preventing each of one or more bus masters in the computer from accessing the memory;

resetting each of the one or more central processing units;

allowing one central processing unit to access the memory and execute a trusted core initialization process; and

after execution of the trusted core initialization process, allowing any other central processing units and any of the one or more bus masters to access the memory.

30. A method as recited in claim 29, further comprising:

mapping a central processing unit reset vector to an initialization vector;

receiving a read request corresponding to the central processing unit reset vector from the one central processing unit;

returning, in response to the read request, the initialization vector to the one central processing unit; and

allowing the one central processing unit to access the memory beginning with the initialization vector.

31. A method as recited in claim 30, wherein the initialization vector is an address within the trusted code in the memory.

32. A method as recited in claim 30, further comprising:

re-mapping the central processing unit reset vector to an additional central processing unit start vector after returning the initialization vector to the one central processing unit; and

returning, in response to any other read request corresponding to the central processing unit reset vector from another central processing unit, the additional central processing unit start vector.

33. A method as recited in claim 32, wherein the initialization vector is an address within the trusted code in the memory and wherein the additional central processing unit start vector and the initialization vector are different addresses within the trusted code in the memory.

34. A method as recited in claim 23, wherein the loading the trusted core comprises copying different portions of the trusted core from a plurality of different sources including one or more of: a local mass storage device, a remote device, and a local chipset.

35. One or more computer-readable memories containing a computer program that is executable by a processor to perform the method recited in claim 23.

36. A method comprising:

allowing a computer to begin operation based on untrustworthy code;

loading, under the control of the untrustworthy code, additional code into memory; and

initiating execution of the additional code in a secure manner despite the untrustworthy code in the computer.

37. A method as recited in claim 36, wherein the initiating further comprises initiating execution of the additional code in a secure manner despite both the untrustworthy code in the computer and other pre-existent state of the computer.

38. A method as recited in claim 36, wherein the initiating execution of the additional code in a secure manner comprises:

preventing each of one or more central processing units in the computer from accessing the memory;

preventing each of one or more bus masters in the computer from accessing the memory;

resetting each of the one or more central processing units;

allowing one central processing unit to access the memory and execute a code initialization process; and

after execution of the code initialization process, allowing any other central processing units and any of the one or more bus masters to access the memory.

39. A method as recited in claim 36, wherein the initiating comprises initiating execution of the additional code in a secure manner without requiring any additional bus transactions to be supported by a processor in the computer.

40. A method as recited in claim 36, further comprising:

mapping a central processing unit reset vector to an initialization vector;

receiving a read request corresponding to the central processing unit reset vector from the one central processing unit;

returning, in response to the read request, the initialization vector to the one central processing unit; and

allowing the one central processing unit to access the memory beginning with the initialization vector.

41. A method as recited in claim 40, further comprising:

re-mapping the central processing unit reset vector to an additional central processing unit start vector after returning the initialization vector to the one central processing unit; and

returning, in response to any other read request corresponding to the central processing unit reset vector from another central processing unit, the additional central processing unit start vector.

42. A method as recited in claim 36, further comprising:

remapping the trusted core to appear at an address where a central processing unit starts executing after being reset.

43. A method as recited in claim 36, further comprising:

receiving, from a central processing unit, a read request corresponding to a central processing unit reset vector;

responding to the read request with instructions to cause the central processing unit to jump to a starting location of the trusted core.

44. A method as recited in claim 36, wherein the loading the additional code comprises copying different portions of the additional code from a plurality of different sources including one or more of: a local mass storage device, a remote device, and a local chipset.

45. One or more computer-readable memories containing a computer program that is executable by a processor to perform the method recited in claim 36.

46. A memory controller comprising:

a first interface to allow communication with a processor;

a second interface to allow communication with a system memory; and

a controller, coupled to the first interface and the second interface, to reset a processor and to allow the processor to execute a code initialization process while preventing any other processors from accessing the system memory.

47. A memory controller as recited in claim 46, wherein the memory controller is included in a processor.

48. A memory controller as recited in claim 46, wherein the first interface comprises a processor bus interface.

49. A memory controller as recited in claim 48, wherein the memory controller operates without requiring the processor bus interface to support any additional commands on the processor bus.

50. A memory controller as recited in claim 46, wherein the system memory comprises a dynamic random access memory.

51. A memory controller as recited in claim 46, wherein the controller is further to allow the processor to execute the code initialization process while preventing any bus masters from accessing the system memory.

52. A memory controller as recited in claim 46, wherein the controller is further to:

reset any other processor coupled to the memory controller prior to allowing the processor to execute the code initialization process;

prevent any other processor and any bus master coupled to the memory controller from accessing the system memory until the one process executes the code initialization process; and

after execution of the code initialization process, allow any other central processing units coupled to the memory controller and any bus masters coupled to the memory controller to access the memory.

53. A memory controller as recited in claim 46, wherein the controller is further to:

map a processor reset vector to an initialization vector;

receive a read request corresponding to the processor reset vector from the processor;

return, in response to the read request, the initialization vector to the processor; and

allow the processor to access the memory beginning with the initialization vector.

54. A memory controller as recited in claim 53, wherein the initialization vector is an address within the code initialization process.

55. A memory controller as recited in claim 53, wherein the controller is further to:

re-map the processor reset vector to an additional processor start vector after returning the initialization vector to the processor; and

return, in response to any other read request corresponding to the processor reset vector from another processor, the additional processor start vector.

56. A memory controller as recited in claim 55, wherein the initialization vector is an address within the code initialization process and wherein the additional processor start vector and the initialization vector are different addresses within the code initialization process.

57. An apparatus comprising:

a processor reset portion to assert a reset signal to a processor; and

a memory protector portion to prevent any bus master from accessing memory until the processor completes execution of a trusted core initialization process.

58. An apparatus as recited in claim 57, wherein the apparatus comprises a programmable logic device.

59. An apparatus as recited in claim 57, wherein the processor reset portion comprises a processor bus interface.

60. An apparatus as recited in claim 57, wherein the memory protector portion comprises a control logic that ignores any request to access the memory received from any bus master.

61. An apparatus as recited in claim 57, further comprising a controller, coupled to the memory protector portion, to prevent another processor from accessing memory until the processor completes execution of the trusted core initialization process.

62. An apparatus as recited in claim 57, further comprising a controller, coupled to the memory protector portion, to:

map a processor reset vector to an initialization vector;

receive a read request corresponding to the processor reset vector from the processor;

return, in response to the read request, the initialization vector to the processor; and

allow the processor to access the memory beginning with the initialization vector.

63. An apparatus as recited in claim 62, wherein the controller is further to:

re-map the processor reset vector to an additional processor start vector after returning the initialization vector to the processor; and

return, in response to another read request corresponding to the processor reset vector from another processor, the additional processor start vector.

64. An apparatus as recited in claim 57, further comprising a storage portion in which a portion of the trusted core is stored.

65. An apparatus as recited in claim 64, wherein the portion of the trusted core stored in the storage portion comprises a platform trusted core portion.

66. A computer comprising:

a processor;

a bus master;

a system memory; and

a memory controller coupled to the processor, the bus master, and the system memory, the memory controller being configured to,

allow access to the system memory from the processor and the bus master operating based on untrustworthy code,

reset the processor to begin a trusted core initialization process, and

prevent the bus master from accessing the system memory until after the trusted core initialization process is completed.

67. A computer as recited in claim 66, further comprising a plurality of additional processors and preventing the plurality of additional processors from accessing the system memory until after the trusted core initialization process is completed.

68. A method comprising:

allowing execution of different trusted cores in a computer to be initiated serially without requiring the computer to be re-booted.

69. A method as recited in claim 68, wherein the allowing further comprises allowing execution of the different trusted cores to be initiated at arbitrary times.

70. A method as recited in claim 68, wherein the different trusted cores are different versions of the same trusted core.

add A1